LoopDelta: Embedding Locality-aware Opportunistic Delta Compression in Inline Deduplication for Highly Efficient Data Reduction

Yucheng Zhang *Nanchang University*
Hong Jiang *University of Texas at Arlington*
Dan Feng *Huazhong University of Science and Technology*
Nan Jiang *East China Jiaotong University*
Taorong Qiu *Nanchang University*
Wei Huang *Nanchang University*

Background

- Redundant data in backup systems

- Data deduplication
  - Removing duplicate chunks

- Delta compression
  - Removing redundancy among similar chunks

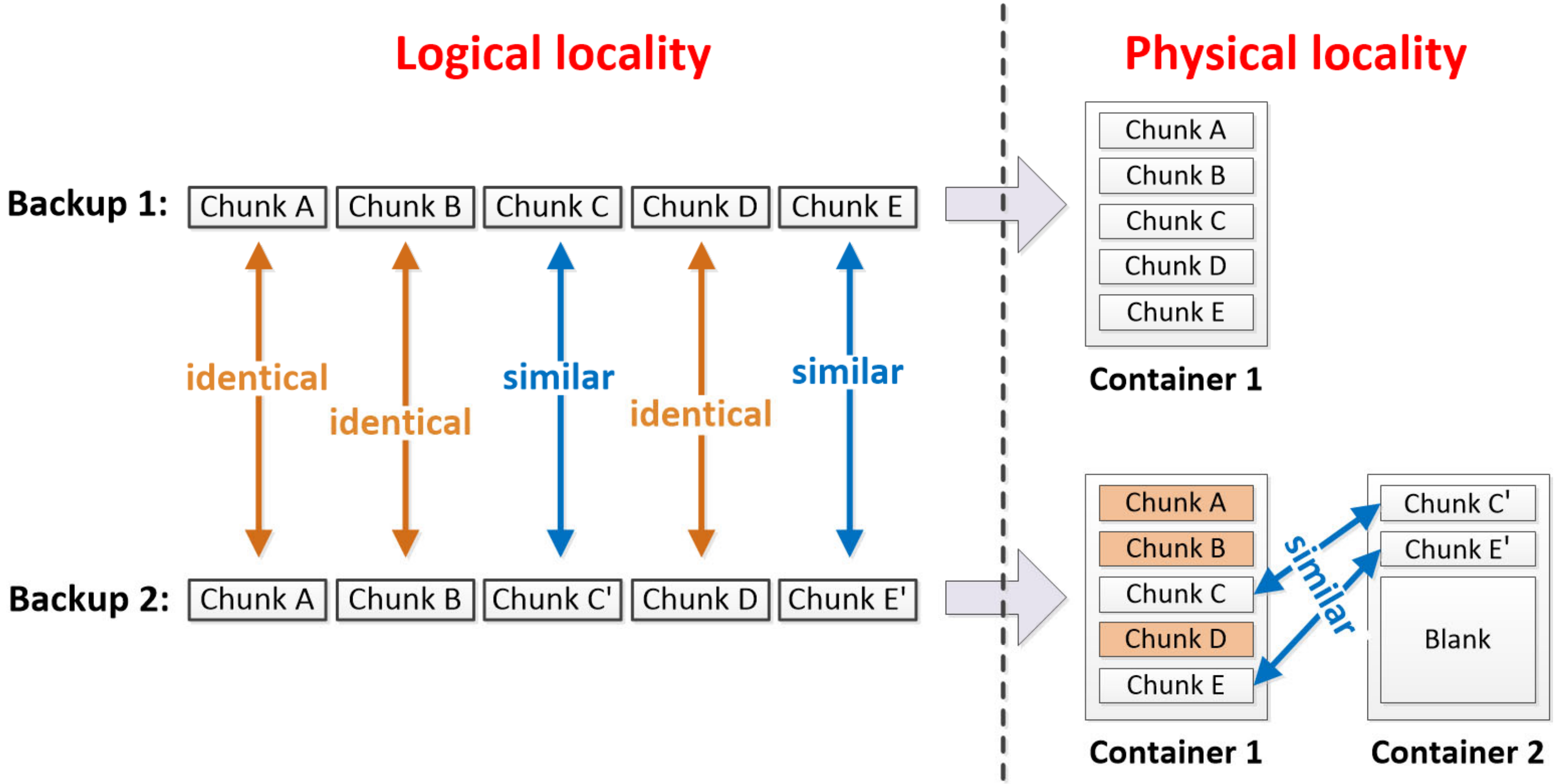
Challenges of adding delta compression to deduplication systems

- Low compression ratio
- Low backup throughput
- Low restore performance
- Missing potential similar chunks when rewriting techniques are applied

Challenge 1: low compression ratio

Redundancy Locality: the repeating patterns of the redundant data among consecutive backups

- Logical Locality: the repeating pattern **before** deduplication
- Physical Locality: the repeating pattern **after** deduplication



Logical locality
Physical locality
Backup 1:
Chunk A
Chunk B
Chunk C
Chunk D
Chunk E
identical
identical
similar
identical
similar
Backup 2:
Chunk A
Chunk B
Chunk C'
Chunk D
Chunk E'
Container 1
Chunk C'
Chunk E'
Blank
similar
Container 1
Container 2

Challenge 1: low compression ratio

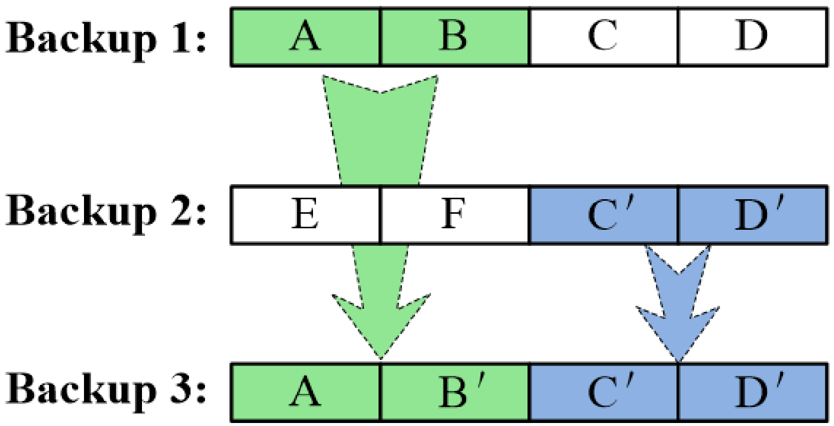
Sketch indexing techniques:

- Logical-locality-based indexing: sketches of the data chunks of the last backup
- Physical-locality-based indexing: sketches of the data chunks stored along with duplicate chunks
- Full indexing: sketches of all data chunks in the backup storage

Logical-locality-based sketch indexing

Disadvantage: Missing potential similar chunks across backup versions.

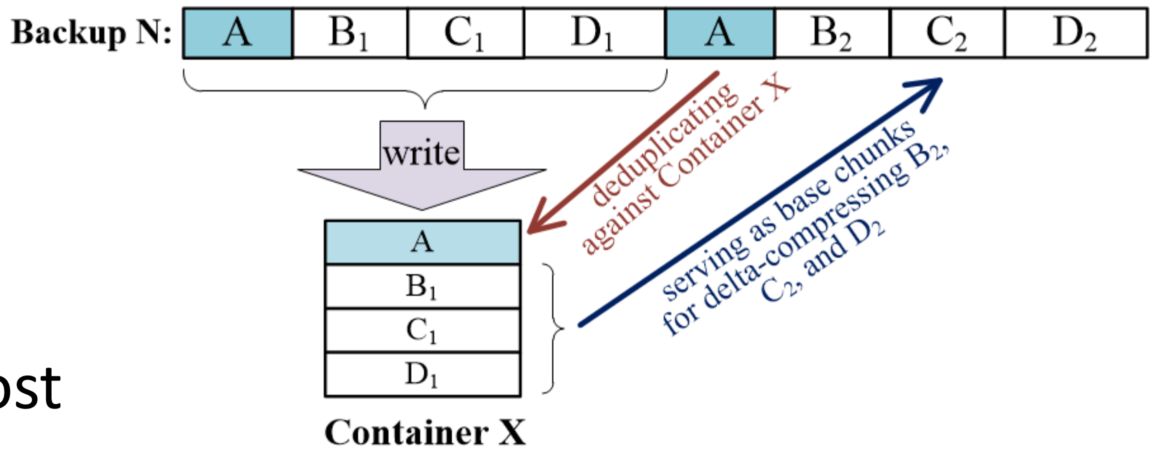
Advantage: high similarity of detected similar chunks.



The best base chunk for delta-compressing a chunk is often its previous copy in the last backup.

Physical-locality-based sketch indexing

Disadvantage: Detecting self-referenced similar chunks as base chunks



Advantage: Detecting most of potential similar chunks

similar chunks from the previous backups **>** self-referenced similar chunks

Full sketch indexing

Disadvantage: Detecting self-referenced similar chunks as base chunks

Advantage: Detecting all potential similar chunks

Upper bound for compression evaluations

Challenge 1: low compression ratio

Complementary capabilities

	Advantage	Disadvantage
Logical locality	High similarity	Missing similar chunks
Physical locality	Detecting almost all similar chunks	Low similarity

Combining the Best of Both Worlds

Dual-locality-based Sketch Indexing: detecting similar chunks by exploiting both logical and physical locality

Challenge 2: low backup throughput

Extra I/Os for reading base chunks on the write path significantly decrease the backup throughput.

Observations:

- Routine operations: accessing containers during deduplication
- Most of the containers holding similar chunks would be accessed during deduplication

Challenge 2: low backup throughput

Observations:

- Routine operations: accessing containers during deduplication
- Most of the containers holding similar chunks would be accessed during deduplication

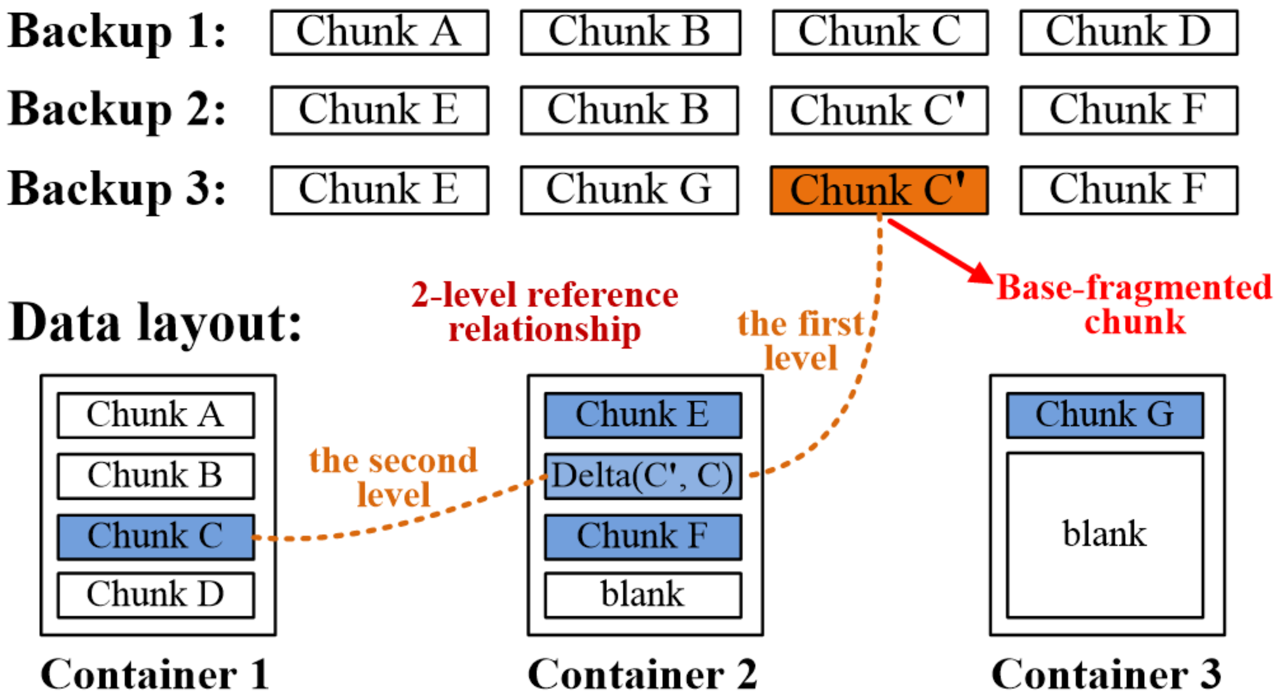
Locality-aware Prefetching:

- Prefetching potential base chunks by piggybacking on routine operations for prefetching metadata during deduplication.

Challenge 3: low restore performance

Extra I/Os for reading base chunks on the read path significantly decrease the restore performance.

- Locality-aware prefetching reduces extra I/Os during restore.
- **Base-fragmented chunks**: Data chunks that refer to deltas whose base chunks requier extra I/Os during restore.



Challenge: Obtaining the container ID of the base chunk of a delta in the system

Challenge 3: low restore performance

Cache-aware Filter:

- Storing fingerprints of base chunks of deltas along with deltas.
- Identifying base-fragmented chunks with the assistance of recently prefetched metadata during deduplication
- rewriting base-fragmented chunks to prevent extra I/Os for base chunks during restore

Challenge 4: missing base chunks

- The rewriting techniques declare infrequently reused containers.
- Base chunks are required during restore.
- Similar chunks detected from infrequently reused containers cannot serve as base chunks.

Challenge 4: missing base chunks

Observations:

Delta compression can be viewed as a two-step process.

- **Step 1**: encoding the target chunk relative a similar chunk and generating a delta
- **Step 2**: removing the target chunk and storing the delta to achieve a data reduction

The target chunk refers to a chunk being backed up, while the similar chunk refers to a chunk in the backup storage.

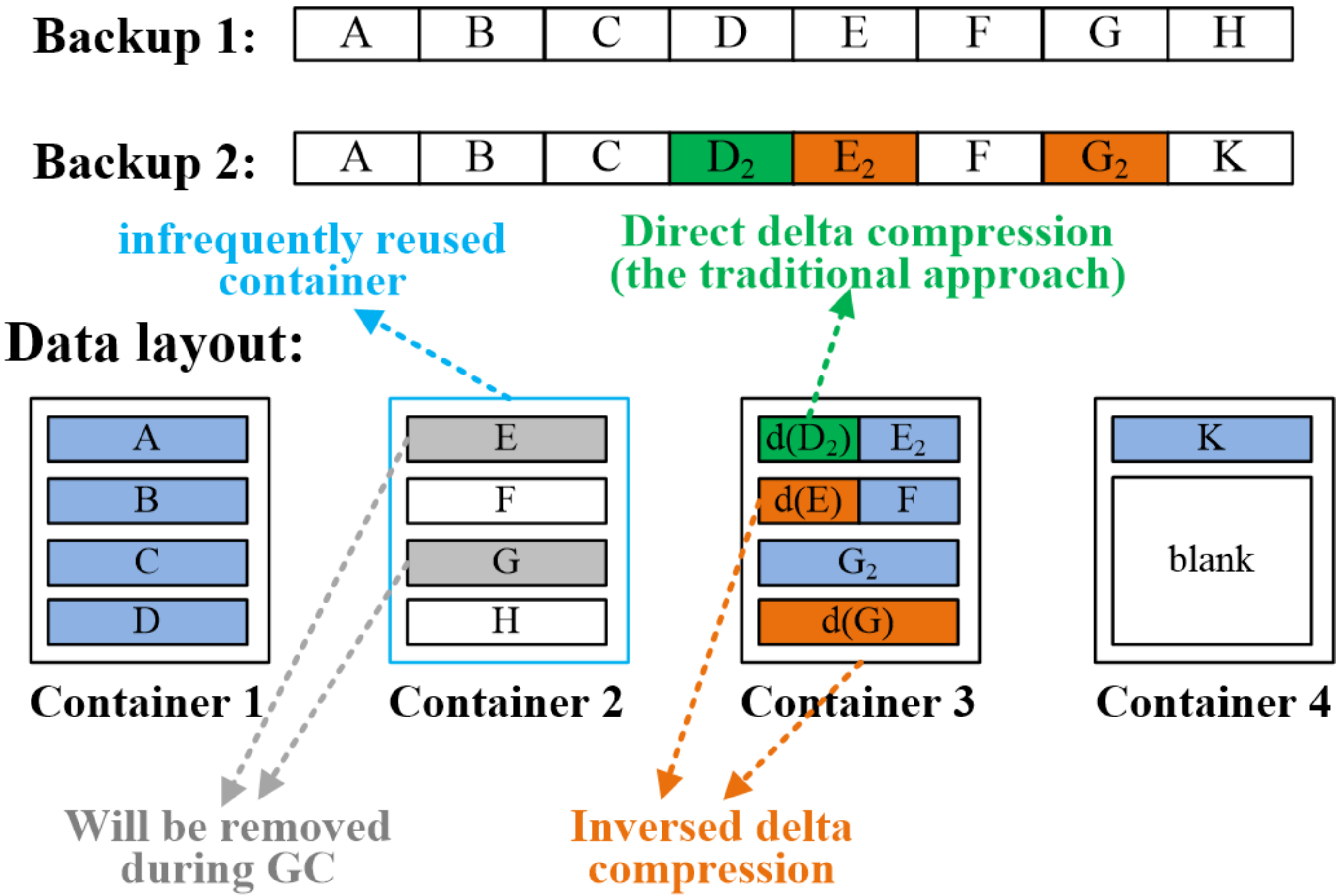
Challenge 4: missing base chunks

Inversed Delta Compression:

Changing the target of delta compression to the chunk in the backup storage.

ATC'23-LoopDelta

- **Step 1**: encoding the detected similar chunk (say, *S*) relative to the chunk (say, *C*) being backed up and generating a delta, storing the delta along with *C*.
- **Step 2**: removing *S* during **Garbage Collection (GC)** to achieve a data reduction.



Backup 1:
A B C D E F G H
Backup 2:
A B C D2 E2 F G2 K
infrequently reused container
Direct delta compression (the traditional approach)
Data layout:
A B C D
Container 1
E F G H
Container 2
d(D2) E2 d(E) F G2 d(G)
Container 3
K blank
Container 4
Will be removed during GC
Inversed delta compression

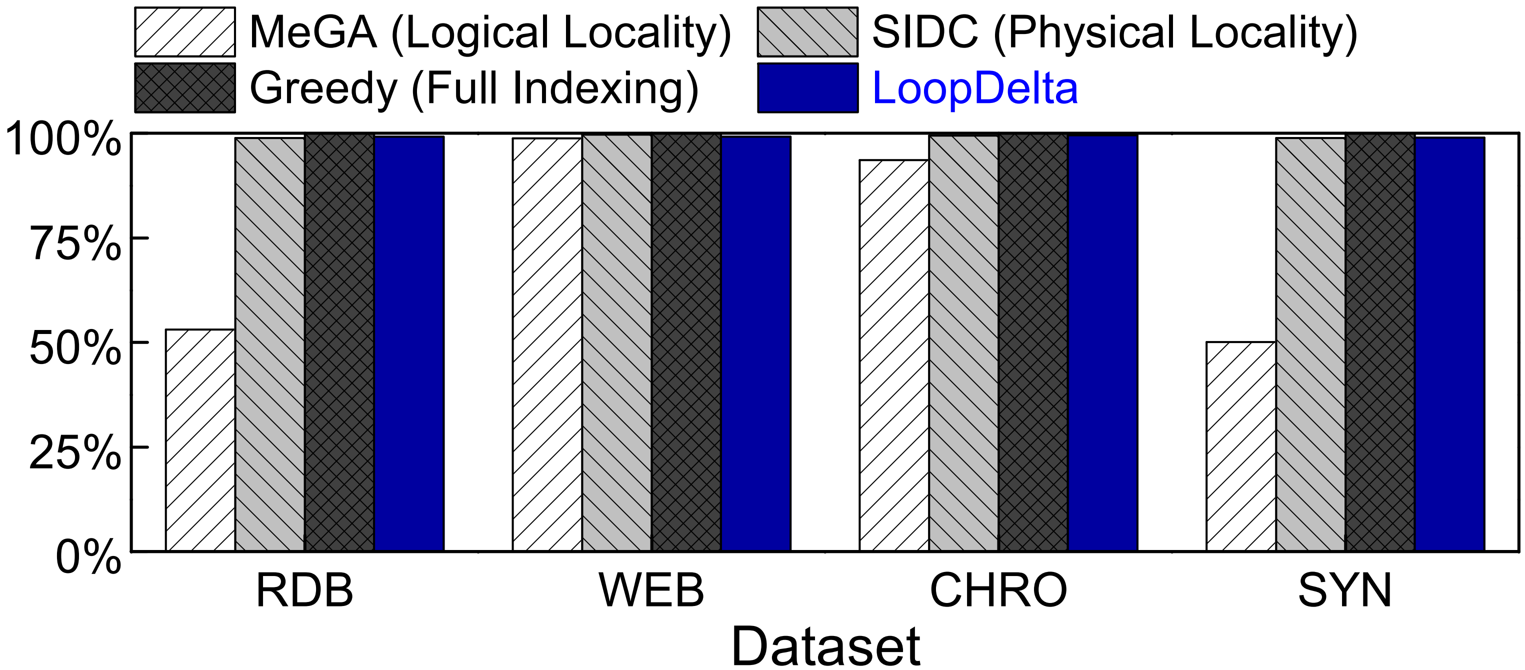
Our approach: LoopDelta

- **Dual-locality-based Sketch Indexing**: low compression ratio
- **Locality-aware Prefetching**: low backup throughput due to extra I/Os for base chunks on the write path
- **Cache-aware Filter**: low restore performance caused by extra I/Os for base chunks on the read path
- **Inversed Delta Compression**: delta compression prohibited by rewriting techniques

Evaluation: datasets

Name	Size	Workload descriptions	Key property
RDB	1080 GB	200 backups of the redis key-value store database	Multi-version inheritance
WEB	330 GB	120 days' snapshots of the website: news.sina.com.	Self-referenced similar chunks
CHM	284 GB	100 versions of source codes of Chromium project from v84.0.4110 to v86.0.4215	
SYN	335 GB	180 synthetic backups by simulating file create/delete/modify operations	Multi-version inheritance

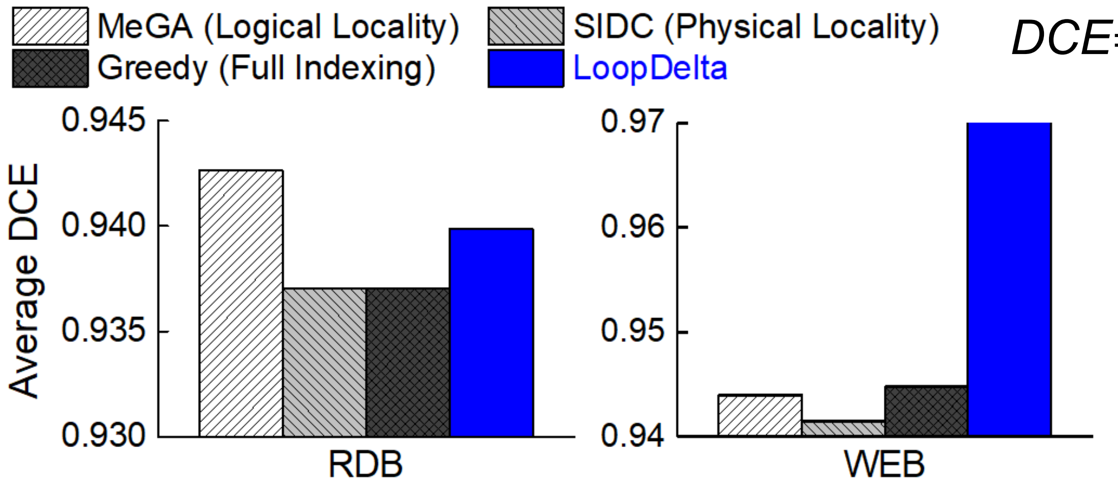
Evaluation: % of detected similar chunks



LoopDelta (our approach) can detect nearly all potential similar chunks.

Evaluation: similarity of detected chunks

A larger value of DCE indicates higher similarity



$$DCE = \frac{size\ of\ delta\text{-}compressed\ bytes}{chunk\ size\ before\ delta\ compression}$$

On dataset (WEB) containing self-referenced similar chunks, our approach detects similar chunks with higher similarity than other approaches.

Evaluation: efficiency of Cache-aware Filter

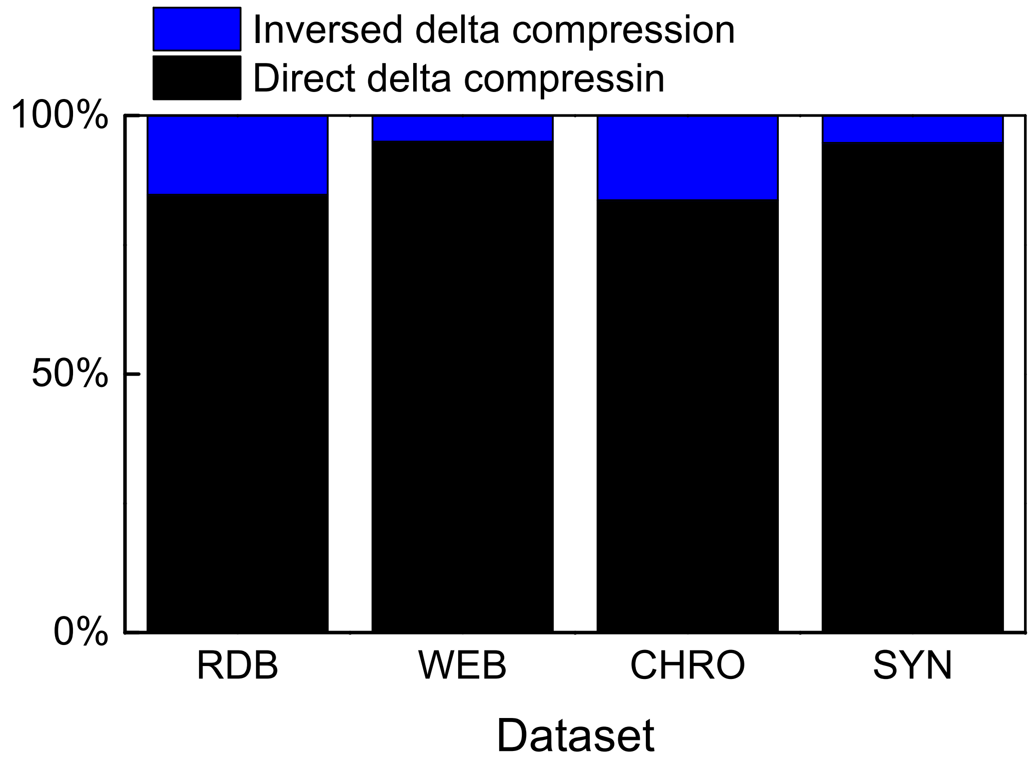
Improvement in restore performance achieved by Cache-aware Filter when rewriting is applied

Dataset	Improvement (%)
RDB	50.6%
WEB	11.7%
CHRO	33.3%
SYN	47.8%

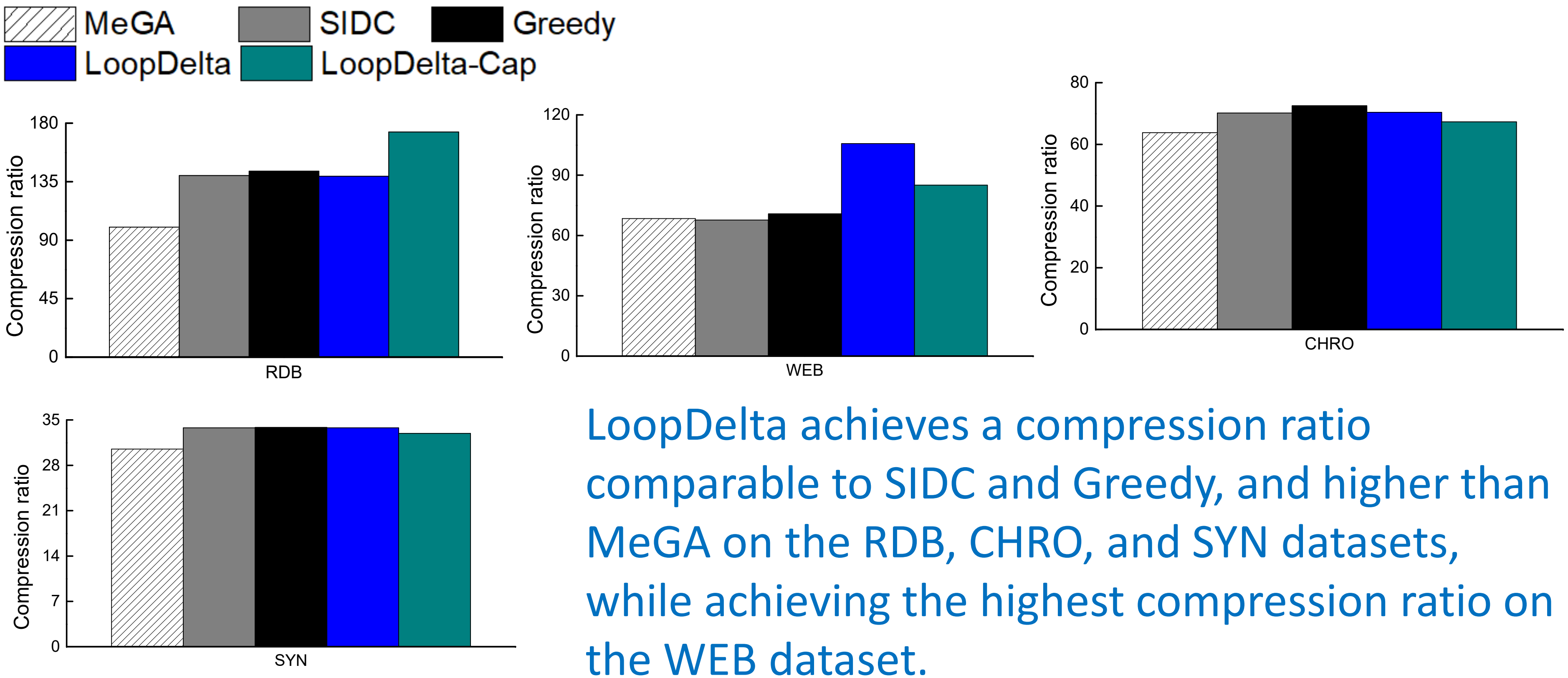
Evaluation: efficiency of Inversed Delta Compression

The rewriting scheme is Capping.

The compression gain: 15.3%, 5%, 16.4%, and 5.3%

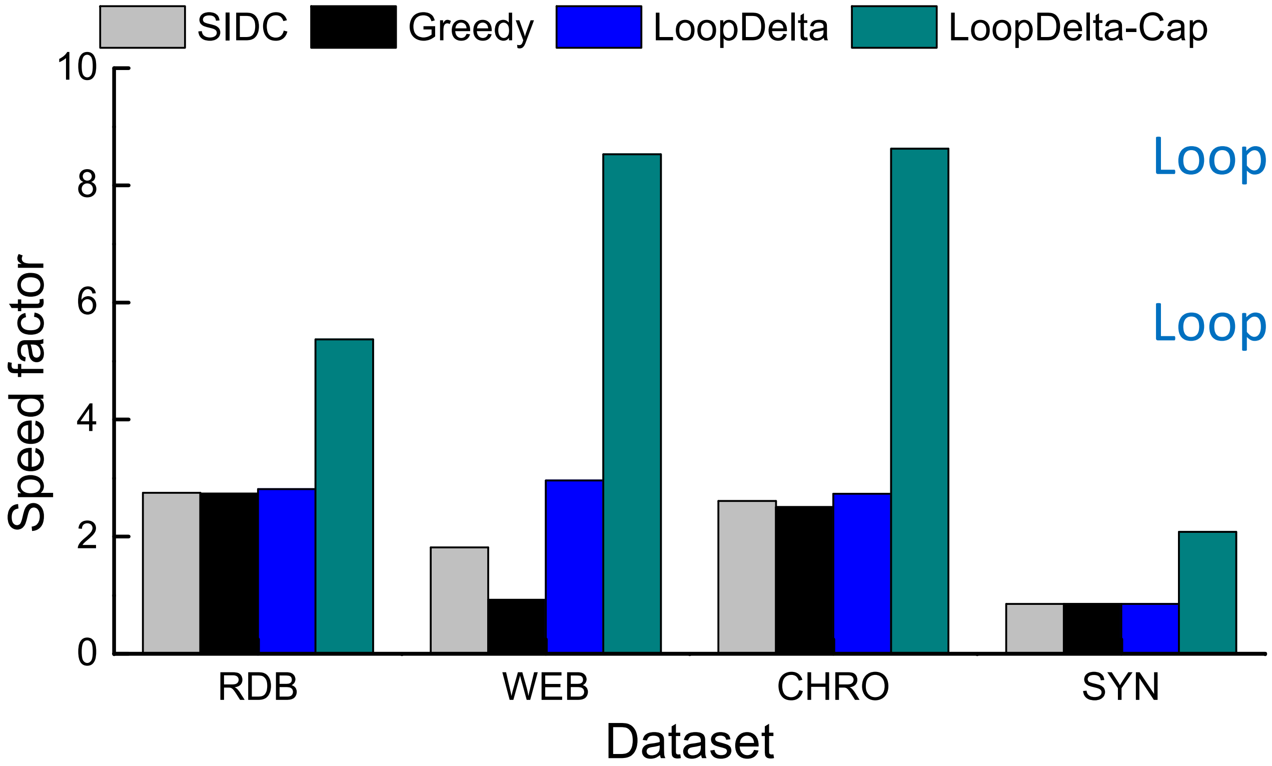


Evaluation: compression ratio



LoopDelta achieves a compression ratio comparable to SIDC and Greedy, and higher than MeGA on the RDB, CHRO, and SYN datasets, while achieving the highest compression ratio on the WEB dataset.

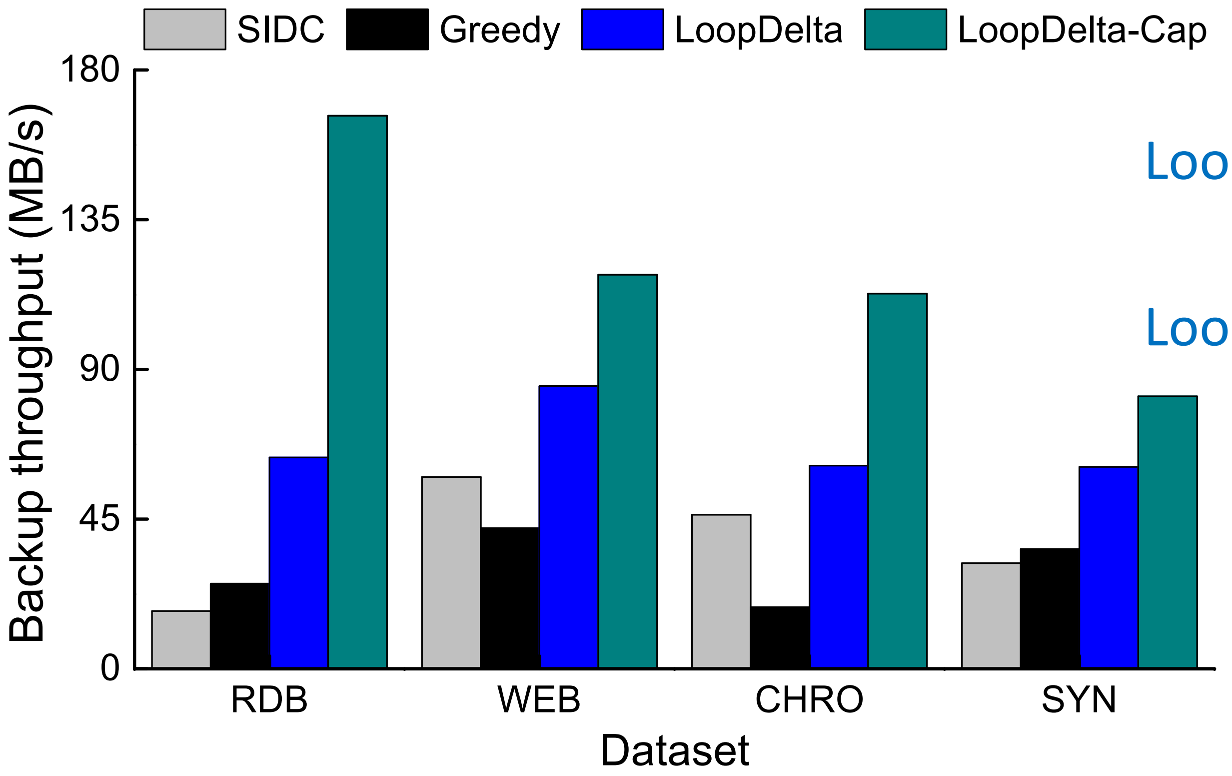
Evaluation: restore performance



LoopDelta > SIDC, Greedy

LoopDelta-Cap > LoopDelta

Evaluation: backup throughput



LoopDelta > SIDC, Greedy

LoopDelta-Cap > LoopDelta

Thank you!

For any inquiries, please email me at
zhangyc_hust@126.com